

## **REMARKS**

Applicant is in receipt of the Office Action mailed October 16, 2007. Claims 43, 45, 49, 50, 52, 53, 58, 59, and 66-68 have been amended. Claim 51 has been cancelled. Claims 43-50 and 52-68 are pending in the case. Reconsideration of the present case is earnestly requested in light of the following remarks.

### **Claim Amendments**

Independent claims 43, 52, 53, and 58 have been amended to include the subject matter of claim 51 (now cancelled).

Additionally, various one of the claims have been amended as indicated above to correct informalities and/or to clarify the intended scope of the claims.

### **Objections**

Claims 52 was objected to for a minor informality, and has been amended accordingly. Applicant thus respectfully requests removal of the objection to this claim.

### **Section 102 Rejections**

Claims 43-68 were rejected under 35 U.S.C. 102(b) as being anticipated by Kudukoli (US Patent Pub. No. 2001/0024211 A1). Applicant respectfully traverses the rejection.

Amended claim 43 recites:

43. A computer-accessible memory medium that stores program instructions executable by a processor to perform:

displaying a node in a graphical program;  
style="padding-left: 40px;">receiving first user input invoking display of a plurality of functions for the node;  
style="padding-left: 40px;">displaying the plurality of functions for the node in response to the first user input;  
style="padding-left: 40px;">receiving second user input selecting a function from the plurality of functions;

determining graphical program code based on the second user input, wherein the determined graphical program code is executable to provide functionality in accordance with the selected function;

associating the determined graphical program code with the node, wherein, when the node in the graphical program executes, the determined graphical program code executes to provide the functionality in accordance with the selected function;

wherein the node represents a subprogram, wherein the program instructions are further executable to perform:

receiving user input invoking expansion of the node; and

displaying the subprogram in response to said invoking.

Nowhere does Kudukoli teach or suggest **wherein the node represents a subprogram, wherein the program instructions are further executable to perform; receiving user input invoking expansion of the node; and displaying the subprogram in response to said invoking**, as recited in amended claim 43.

Cited paragraphs [0217]-[0221] (from the Examiner's arguments directed to claim 51, now cancelled) describe various inputs to a New VI Object Reference node that specify various attributes of an *object* to be created by the New VI Object Reference node, including a "style" input to the New VI Object Reference node that specifies the style or sub-class of object to create, a "position/next to" input to the New VI Object Reference node that specifies a position for the new object, an "error in" input to the New VI Object Reference node that describes error conditions that occur prior to the execution of this function, a "path" input to the New VI Object Reference node that specifies the location of a user control/VI, and a "bounds" input to the New VI Object Reference node that specifies the size of the new object.

Nowhere do these citations (nor Kudukoli in general) describe receiving user input invoking expansion of the node, and displaying a subprogram represented by the node in response, as claimed.

Thus, Kudukoli fails to disclose this feature of claim 43.

Thus, for at least the reasons provided above, Applicant submits that the cited art fails to disclose all the features and limitations of amended claim 43, and so claim 43, and

those claims respectively dependent therefrom, are patentably distinct and non-obvious over the cited art, and are thus allowable.

Independent claims 52, 53, and 58 each also includes similar limitations as claim 43, and so the above arguments apply with equal force to these claims.

Thus, for at least the reasons provided above, Applicant submits that claims 52, 53, and 58, and those claims respectively dependent therefrom, are patentably distinct and non-obvious over the cited art, and are thus allowable.

Applicant also asserts that numerous ones of the dependent claims recite further distinctions over the cited art.

For example, Applicant submits that Kudukoli fails to teach or suggest **changing the first node icon to a second appearance based on the second user input**, wherein said **changing the first node icon to a second appearance includes displaying an image corresponding to the selected function**, as recited in claim 44 (where the first node icon represents the node).

Cited Figure 21 and paragraphs [0217]-[0221] are directed to a New VI Object Reference node that when executed creates a new VI object and outputs a reference to the new VI object. More specifically, Figure 21 illustrates a cascade of menus for specifying the object that the New VI Object Reference node will create upon execution. Applicant notes that the menus actually display various VI objects that can be created by the node, and attributes for the objects, e.g., a VI Object, a Panel Object, or a Diagram Object, and attributes such as data or object type, etc. Applicant thus submits that Kudukoli's node displays objects and object attributes, not functions. More specific to claim 44, nowhere does Figure 21 show changing the appearance of the New VI Object Reference node icon in response to the user selecting a particular function.

Cited paragraph [0217] describes a “style” input to the New VI Object Reference node that specifies the style or sub-class of object to create, and presents an example where the VI object class input is “slide”, and “vertical pointer slide” may be selected as the style of the slide. Applicant respectfully notes that these attributes are for the object to be created by the New VI Object Reference node, not the node itself.

Similarly, cited paragraph [0218] describes a “position/next to” input to the New VI Object Reference node that specifies a position for the new object. Again, this attribute is for the object to be created by the New VI Object Reference node, not the node.

Paragraph [0219] describes an “error in” input to the New VI Object Reference node that describes error conditions that occur prior to the execution of this function; paragraph [0220] describes a “path” input to the New VI Object Reference node that specifies the location of a user control/VI; and paragraph [0221] describes a “bounds” input to the New VI Object Reference node that specifies the size of the new object. As may be seen, none of these citations (nor Kudukoli in general) teaches changing the appearance of the (New VI Object Reference) node icon based on a function for the node selected by the user. More specifically, the citations fail to disclose changing the first node icon to a second appearance based on the second user input (selecting a function), including displaying an image corresponding to the selected function.

Thus, Kudukoli fails to teach or suggest this feature of claim 44, and so claim 44, and those claims dependent therefrom, are patentably distinct and non-obvious over the cited art, and are thus allowable.

Nor does Kudukoli teach or suggest **wherein said changing the first node icon to a second appearance comprises replacing the first node icon with a second node icon**, as recited in claim 45 (where the first node icon represents the node).

As noted above, cited Figure 21 illustrates a cascade of menus for specifying the object that the New VI Object Reference node will create upon execution. Nowhere does Figure 21 indicate changing the appearance of the (New VI Object Reference) node in response to the user selecting a particular function for the node. Applicant respectfully submits that the menus shown in Figure 21 are *not* the icon of the node, an example of which is shown in Figure 13.

Cited paragraphs [0215]-[0216] are directed to input of a New VI Object Reference node. More specifically, paragraph [0215] describes a “vi object class” input to the New VI Object Reference node that specifies the type of object to create, e.g., a “slide” value may be chosen to designate that the reference to obtain is a reference to a

slide user interface control. Note that this input is not germane to the appearance of the (New VI Object Reference) node. Paragraph [0216] describes an “owner reference” input to the New VI Object Reference node that specifies a reference to the VI or VI object that will “own” or “contain” the new object, e.g., the owner may be the VI, and the new object may be a new function node to add. Again, specifying such ownership for the object to be created is not germane to the appearance of the (New VI Object Reference) node.

As may be seen, none of these citations (nor Kudukoli in general) teaches changing the appearance of the (New VI Object Reference) node icon, including replacing the first node icon with a second node icon, based on a function for the node selected by the user.

Thus, Kudukoli fails to teach or suggest this feature of claim 45, and so claim 45, and those claims dependent therefrom, are patentably distinct and non-obvious over the cited art, and are thus allowable.

Nor does Kudukoli teach or suggest **wherein said displaying the plurality of functions for the node in response to the first user input comprises: displaying a plurality of function classes for the node; and in response to user input selecting a function class, displaying the plurality of functions, wherein the plurality of functions are in the selected function class**, as recited in claim 49.

Cited Figure 22 “illustrates how a user may choose a value for the style input by selecting from a hierarchical menu. For example, if ‘slide’ is chosen as the vi object class input, then ‘vertical pointer slide’ may be chosen for the style input.” As with Figure 21, described above, the cascaded menus shown in Figure 22 present selectable attributes for the object to be created by the node, not the (New VI Object Reference) node itself. More specifically, in the example of Figure 22, the selected style for the user-selected Panel Object/Control/Numeric/Slide object is a Vertical Pointer Slide, shown selected in the final menu of the cascade. Again, Applicant notes that the menu items shown are objects that are selectable to be created, and their possible attributes (including the Vertical Pointer Slide image). Nowhere does Figure 22 illustrate displaying a plurality of *function classes* for the node, and in response to user input selecting a *function class*,

displaying the plurality of functions, wherein the plurality of functions are in the selected function class, as claimed. Applicant respectfully submits that *object classes or attributes are not function classes*. On p.5, lines 18-19, Applicant's Specification provides examples of function classes: "read, write, timing, and triggering". Nowhere does Figure 22 (nor Kudukoli in general) disclose such function classes, nor selecting a function for a node by selecting from displayed function classes, then selecting a function from a plurality of displayed functions of the selected function class.

As discussed in detail above, cited paragraphs [0217]-[0221] describe various inputs to a New VI Object Reference node that specify various attributes of an object to be created by the New VI Object Reference node, including a "style" input to the that specifies the style or sub-class of object to create, a "position/next to" input to the New VI Object Reference node that specifies a position for the new object, an "error in" input to the New VI Object Reference node that describes error conditions that occur prior to the execution of this function, a "path" input to the New VI Object Reference node that specifies the location of a user control/VI, and a "bounds" input to the New VI Object Reference node that specifies the size of the new object.

None of these citations (nor Kudukoli in general) describes or even hints at selecting a function for a node via selection of a function class, then selecting a function from that function class as claimed.

Thus, Kudukoli fails to teach or suggest these features of claim 49, and so claim 49, and those claims dependent therefrom, are patentably distinct and non-obvious over the cited art, and are thus allowable.

Applicant respectfully submits that Kudukoli also fails to teach or suggest all the features and limitations of claim 50.

For example, Kudukoli fails to teach or suggest **wherein the node is a data acquisition (DAQ) node**, as recited in claim 50.

Cited Figure 7 and paragraphs [0146]-[0147] are directed to "an example in which a graphical program has been programmatically generated based on a state diagram." More specifically, as stated in paragraph [0146], "FIG. 7 illustrates an exemplary graphical user interface of a state diagram editor program, in which the state diagram

used in programmatically generating the graphical program is displayed. FIG. 7 also illustrates a block diagram portion corresponding to the ‘Number is NOT Prime’ state of the state diagram.”

Paragraph [0147] discusses maintaining an association between a generated graphical program and the received program information used in generating and/or modifying the graphical program to facilitate automatically updating the generated program when the program information is modified.

Nowhere do these citations disclose a DAQ node with the functionality recited in claim 1. Thus, Kudukoli fails to teach or suggest this feature of claim 50.

Nor does Kudukoli disclose **wherein the plurality of functions for the node comprise a plurality of DAQ functions**, as recited in claim 50. Applicant respectfully notes that Kudukoli never even mentions DAQ functions. Nor does cited Figure 7 illustrate a plurality of DAQ functions. Thus, Kudukoli does not, and cannot, disclose displaying a plurality of DAQ functions for a DAQ node, as claimed. Thus, Kudukoli fails to teach this feature of claim 50.

Nor does Kudukoli disclose **wherein, prior to said associating, the DAQ node comprises one of:**

- a generic read node;**
- a generic write node;**
- a generic channel creation node;**
- a generic timing node; or**
- a generic triggering node; and**

**wherein, after said associating, the DAQ node comprises one of:**

- a specific read node in accordance with the selected function;**
- a specific write node in accordance with the selected function;**
- a specific channel creation node in accordance with the selected function;**
- a specific timing node in accordance with the selected function; or**
- a specific triggering node in accordance with the selected function,** as recited in claim 50.

As discussed above, Figure 7 and paragraphs [0146]-[0147] are directed to “an example in which a graphical program has been programmatically generated based on a state diagram.” More specifically, as stated in paragraph [0146], “FIG. 7 illustrates an exemplary graphical user interface of a state diagram editor program, in which the state diagram used in programmatically generating the graphical program is displayed. FIG. 7 also illustrates a block diagram portion corresponding to the ‘Number is NOT Prime’ state of the state diagram.” Paragraph [0147] discusses maintaining an association between a generated graphical program and the received program information used in generating and/or modifying the graphical program to facilitate automatically updating the generated program when the program information is modified.

Nowhere does Kudukoli disclose a generic DAQ node, e.g., a generic read node, a generic write node, a generic channel creation node, a generic timing node, or a generic triggering node, being converted to a specific DAQ node, e.g., a specific read node, a specific write node, a specific channel creation node, a specific timing node, or a specific triggering node, in accordance with the selected function, and in response to associating determined graphical program code with the DAQ node.

Thus, Kudukoli fails to disclose these features of claim 50.

Thus, Kudukoli fails to teach or suggest all the features and limitations of claim 50, and so claim 50, and those claims dependent therefrom, are patentably distinct and non-obvious over the cited art, and are thus allowable.

Amended independent claim 59 recites:

59. A computer-accessible memory medium that stores program instructions executable by a processor to perform:

displaying a node in a graphical program, wherein the node has a first node icon which is displayed in the graphical program, and wherein the first node icon has a first appearance;

wiring one or more inputs to the node, wherein the one or more inputs have respective one or more data types;

determining a function from a plurality of possible functions for the node based on the one or more data types;

determining graphical program code based on the determined function, wherein the determined graphical program code is executable to perform the determined function; and

associating the determined graphical program code with the node, wherein, when the node executes in the graphical program, the determined graphical program code is operable to execute to perform the determined function.

Nowhere does Kudukoli disclose **wiring one or more inputs to the node, wherein the one or more inputs have respective one or more data types and determining a function from a plurality of possible functions for the node based on the one or more data types**, as recited in claim 59

Cited Figure 13 and paragraphs [0212]-[0221] are directed to a New VI Object Reference node that when executed creates a new VI object and outputs a reference to the new VI object. More specifically, Figure 13 illustrates inputs and outputs of the New VI Object Reference node, and paragraphs [0212]-[0221] describe these inputs of the New VI Object Reference node. For example, paragraph [0214] describes an “auto wire” input to the New VI Object Reference node that specifies whether to automatically wire the terminals of the new object with compatible terminals of a source object. Paragraph [0215] describes a “vi object class” input to the New VI Object Reference node that specifies the type of object to create, e.g., a “slide” value may be chosen to designate that the reference to obtain is a reference to a slide user interface control. Paragraph [0216] describes an “owner reference” input to the New VI Object Reference node that specifies a reference to the VI or VI object that will “own” or “contain” the new object, e.g., the owner may be the VI, and the new object may be a new function node to add. Similarly, paragraph [0217]-[0221] describe various further inputs to the New VI Object Reference node, including a “style” input to the New VI Object Reference node that specifies the style or sub-class of object to create, a “position/next to” input to the New VI Object Reference node that specifies a position for the new object, an “error in” input to the New VI Object Reference node that describes error conditions that occur prior to the execution of this function, a “path” input to the New VI Object Reference node that specifies the

location of a user control/VI, and a “bounds” input to the New VI Object Reference node that specifies the size of the new object.

Applicant respectfully notes that Kudukoli’s New VI Object Reference node creates a specified object based on the values of the input data, i.e., the user specifies precisely what object is to be created, along with any attributes needed to specify creation of the object, and the node creates the object accordingly. In direct contrast, in claim 59, the function to be performed by the node is determined based on the *data types* of the inputs. This functionality is referred to in the programming arts as *polymorphism*, and is neither taught nor suggested by Kudukoli.

Thus, Applicant submits that Kudukoli fails to disclose all the features of claim 59, and so claim 59, and those claims respectively dependent therefrom, are patentably distinct and non-obvious over the cited art, and are thus allowable.

Independent claims 66, 67, and 68 each includes similar features and limitations as claim 59, and so the above arguments apply with equal force to these claims. Thus, for at least the reasons provided above, claims 66, 67, and 68, and those claims respectively dependent therefrom, are patentably distinct and non-obvious over the cited art, and are thus allowable.

Applicant also asserts that numerous other ones of the dependent claims recite further distinctions over Kudukoli. However, since the independent claims have been shown to be patentably distinct, a further discussion of the dependent claims with respect to Kudukoli is not necessary at this time.

Removal of the section 102 rejection of claims 43-50, and 52-68 is earnestly requested.

Claims 43, 52-53, 58-59, and 66-68 were rejected under 35 U.S.C. 102(b) as being anticipated by Zhang, et al. (US Patent No. 6,282,699, “Zhang”). Applicant respectfully traverses the rejection.

Applicant respectfully notes that in Zhang, textual code, e.g., C, C++, HiQ script code, MatLab script code, etc., is included in or associated with a code node, and the code node is included in a graphical program. The graphical program is compiled and executed, where when the code node executes, the textual code is executed.

Nowhere does Zhang disclose **wherein the node represents a subprogram, wherein the program instructions are further executable to perform: receiving user input invoking expansion of the node; and displaying the subprogram in response to said invoking**, as recited in claim 43.

Applicant respectfully notes that these features of claim 43 were originally presented in claim 51, now cancelled, and that claim 51 was not rejected under Zhang. Thus, Applicant submits that since independent claim 43 now includes this limitation, claim 43 is patentably distinct and non-obvious over Zhang.

Additional arguments for the patentability of claim 43 over Zhang are now provided.

Zhang fails to teach or suggest **receiving first user input invoking display of a plurality of functions for the node, and displaying the plurality of functions for the node in response to the first user input**, as recited in claim 43.

Cited Figures 10-12, col.10:24-51, and col.16 describe various examples of a user entering textual code into a code node, e.g., manually, or via importing a textual code file or script, where the code node is included in a graphical program. As indicated above, the graphical program is compiled and executed, including executing the code node and the textual code included in the code node.

Applicant has carefully reviewed these citations, and Zhang in general, and can find no description of user input invoking display of functions for a node and displaying the plurality of functions in response. Rather, as indicated above and in the citations, in Zhang, the user either enters textual code directly into a code node in a graphical program, or imports a textual code file or script into the code node. For example, Figures 10-12 illustrate specification and configuration of inputs and outputs for a script node (a code node (manually or via import), where the included textual code is a script), col.10:24-51 describes the user's entry of textual code into the code node, and col. 16 describes HiQ and MatLab scripts. Nowhere do these citations describe invocation or display of a plurality of functions for a node in response to user input.

Thus, Zhang does not disclose this feature of claim 43.

Zhang also fails to teach or suggest **receiving second user input selecting a function from the plurality of functions, and determining graphical program code based on the second user input, wherein the determined graphical program code is executable to provide functionality in accordance with the selected function**, as recited in claim 43.

Cited Figure 12 illustrates user-specification and configuration of I/O for a code node, and Figure 13 illustrates a user importing textual code into the code node. Col.10:24-col.11:51 describes the user's entry of textual code into the code node, e.g., manually or via import, and compiling the graphical program, including the textual code or script, to generate executable instructions, e.g., machine code. Col.17 describes creating a HiQ script for the code node, configuring the code node in a graphical program with the script, and executing the graphical program, thereby executing the script.

Again, nowhere do these citations, nor Zhang in general, disclose determining *graphical program code* based on user-selection of a function from a plurality of displayed functions for the node. Rather, as indicated above and in the citations, in Zhang, the user either enters textual code directly into a code node in a graphical program, or imports a textual code file or script into the code node. The graphical program is compiled, and upon execution, the code node executes or invokes execution of the textual code. Thus, Zhang does not disclose this feature of claim 43.

Nor does Zhang disclose **associating the determined graphical program code with the node, wherein, when the node in the graphical program executes, the determined graphical program code executes to provide the functionality in accordance with the selected function**, as recited in claim 43.

Cited Figures 13-14, col.10:53-col.11:56, and col.18 do not disclose these features of claim 43. For example, Figure 13 is a screen shot illustrating a user typing in textual code, such as a script, into a script node, as well as the user importing textual code from another source, Figure 14 is a screen shot illustrating execution of a graphical program including a MatLab script node. Col.10:53-col.11:56 describes the user's entry of a HiQ or MatLab script into the code node, e.g., manually or via import, and compiling the

graphical program, including the textual code or script, to generate executable instructions, e.g., machine code. Col.18 describes creating a MatLab script for the code node, configuring the code node in a graphical program with the script, and executing the graphical program, thereby executing the script, as well as importing/exporting HiQ and MatLab scripts, and configuring a script server to execute the script upon execution of the graphical program, e.g., invoked from the code node.

Applicant can find no mention of associating determined graphical program code with a node, where, when the node in the graphical program executes, the determined graphical program code executes to provide the functionality in accordance with the user-selected function, as claimed. In fact, as noted above, Zhang fails to disclose determining graphical program code at all, and so does not, and cannot, teach this feature.

Thus, for at least the reasons provided above, Applicant submits that Zhang fails to teach or suggest all the features and limitations of claim 43, and so claim 43, and those claims respectively dependent therefrom, are patentably distinct and non-obvious over the cited art, and are thus allowable.

Independent claims 52, 53, and 58 each includes similar limitations as claim 43, and so the above arguments apply with equal force to these claims. Thus, for at least the reasons provided above, claims 52, 53, and 58, and those claims respectively dependent therefrom, are patentably distinct and non-obvious over Zhang, and are thus allowable.

Regarding independent claims 59, and 66-68:

Nowhere does Zhang disclose **wiring one or more inputs to the node, wherein the one or more inputs have respective one or more data types and determining a function from a plurality of possible functions for the node based on the one or more data types**, as recited in claim 59.

Applicant respectfully notes that the functionality of Zhang's code node is determined by textual code entered into the node by the user (e.g., via manual typing or importing a textual file or script). In direct contrast, in claim 59, the function to be performed by the node is determined based on the *data types* of the inputs wired to the

node. This functionality is referred to in the programming arts as *polymorphism*, and is neither taught nor suggested by Zhang.

Thus, Applicant submits that Zhang fails to disclose all the features of claim 59, and so claim 59, and those claims respectively dependent therefrom, are patentably distinct and non-obvious over the cited art, and are thus allowable.

Independent claims 66, 67, and 68 each includes similar features and limitations as claim 59, and so the above arguments apply with equal force to these claims. Thus, for at least the reasons provided above, claims 66, 67, and 68, and those claims respectively dependent therefrom, are patentably distinct and non-obvious over the cited art, and are thus allowable.

Applicant also asserts that numerous ones of the dependent claims recite further distinctions over Zhang. However, since the independent claims have been shown to be patentably distinct, a further discussion of the dependent claims with respect to Zhang is not necessary at this time.

Removal of the section 102 rejection of claims 43, 52-53, 58-59, and 66-68 is earnestly requested.

## **CONCLUSION**

Applicant submits the application is in condition for allowance, and an early notice to that effect is requested.

If any extensions of time (under 37 C.F.R. § 1.136) are necessary to prevent the above-referenced application(s) from becoming abandoned, Applicant(s) hereby petition for such extensions. The Commissioner is hereby authorized to charge any fees which may be required or credit any overpayment to Meyertons, Hood, Kivlin, Kowert & Goetzel P.C., Deposit Account No. 50-1505/5150-80201/JCH.

Also filed herewith are the following items:

- Request for Continued Examination
- Terminal Disclaimer
- Power of Attorney By Assignee and Revocation of Previous Powers
- Notice of Change of Address
- Other:

Respectfully submitted,

/Jeffrey C. Hood/

Jeffrey C. Hood, Reg. #35198  
ATTORNEY FOR APPLICANT(S)

Meyertons, Hood, Kivlin, Kowert & Goetzel PC  
P.O. Box 398  
Austin, TX 78767-0398  
Phone: (512) 853-8800  
Date: 2007-12-12 JCH/MSW